Rapport Projet Java

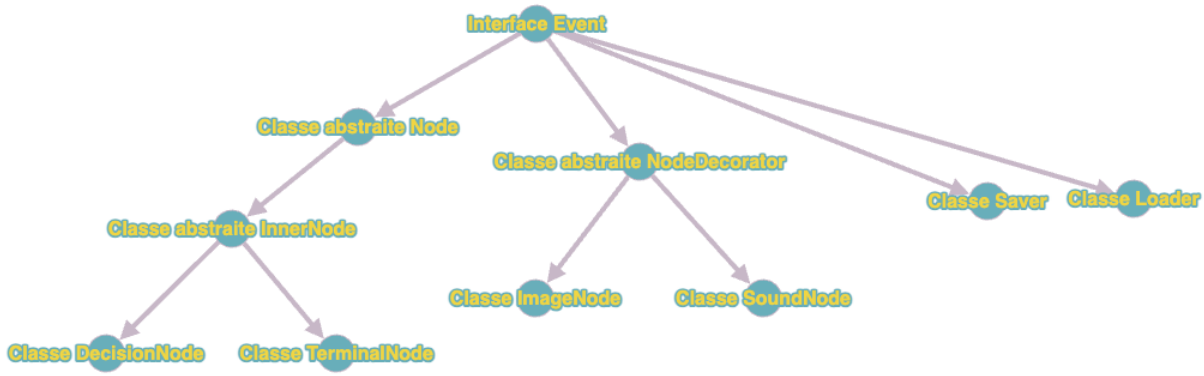
Voici le rapport de notre projet Java, exercice confié par Monsieur Gilbert en 2023 dans le cadre de la matière Java-Objet, en troisième année de licence MIDO. L'objectif de ce projet était de programmer un jeu de type « Vous êtes le Héros », où une histoire évolue en fonctions des choix de l'utilisateur, au sein d'un univers bien précis.

1. Structure du Code et Hiérarchies de Classes

Notre projet est articulé autour d'une architecture de classes hiérarchique, avec une division claire entre les classes principales pour la représentation du jeu, les événements, et l'univers.

Package representation :

Nous avons rigoureusement suivi les directives données dans le projet pour la programmation de l'interface Event et ses dérivés. La seule liberté que nous avons prise dans ce package a été d'ajouter deux classes « Saver » et « Loader », permettant d'enregistrer/charger une partie. Nous avons choisi d'intégrer ces deux fonctionnalités comme des classes de Event car ainsi, à toute étape du jeu, l'utilisateur a la possibilité de passer d'un Event à un Saver/Loader, ou d'un Saver/Loader au prochain nœud. Ces deux fonctionnalités se présentent en effet comme des étapes dans le jeu. Voici la représentation graphique de cette famille.

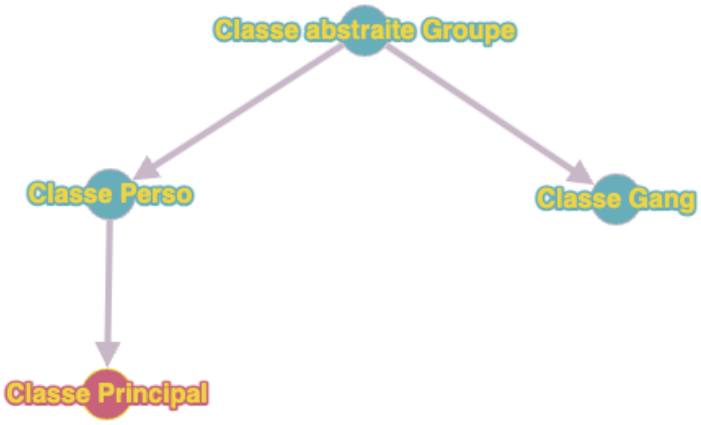


Package Univers :

Le package Univers permet beaucoup plus de liberté, étant moins contraignant au niveau des directives. Nous avons donc pris le temps de bien réfléchir avant de nous lancer dans la construction de ce package.

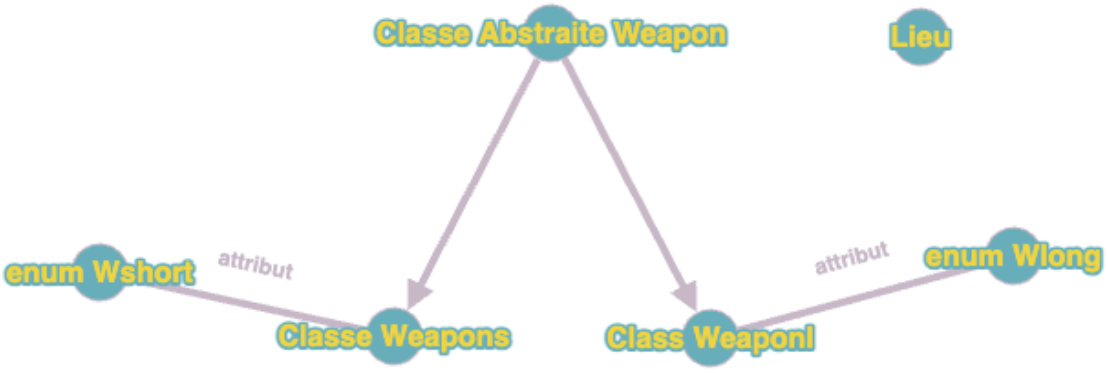
Groupe :
Notre première démarche a été de représenter des personnages, tels que des gangs, des personnages solitaires et le personnage principal. Le personnage principal découlant naturellement des personnages solitaires, nous avons pensé à un lien de parenté. Nous avons également souhaité que le personnage puisse avoir un allié, qu'il soit un personnage

ou un gang. Nous avons ainsi créé une classe mère abstraite Groupe représentant les alliés potentiels, avec des méthodes communes aux gangs et aux personnages. Nous avons décidé de hiérarchiser comme suit.



Armes :

Nous avons ensuite réfléchi à attribuer des armes à nos personnages pour simuler des combats. Nous avons cherché des moyens de complexifier ces affrontements et avons envisagé de créer deux classes d'armes : celles de courte portée et celles de longue portée. Nous avons décidé que les armes de courte portée devaient prévaloir dans des espaces clos, tandis que les armes de longue portée l'emporteraient dans des espaces ouverts. De plus, nous avons logiquement attribué une puissance aux armes de même portée, créant ainsi un enjeu pour l'utilisateur. Celui-ci doit réfléchir à l'endroit où il se trouve et à la puissance de son arme avant de s'engager dans un combat. Si l'utilisateur possède la même arme que son ennemi, nous avons décidé de lui accorder une chance sur deux de gagner. Nous avons ainsi créé deux énumérations et deux classes d'armes selon leur portée, ainsi qu'une classe abstraite mère pour simuler des combats entre deux armes de portée différentes.



Package main

Le package main a pour unique fonction de gérer le déroulement de l'histoire. Il contient une seule classe, « Histoire », qui renferme uniquement la méthode main.

2. Les difficultés rencontrées

Nous avons été confrontés à de nombreuses difficultés d'implémentation et de rédaction tout au long du projet. Heureusement, nous avons su rebondir rapidement et, dans tous les cas, sauf un, nous avons réussi à identifier l'origine de nos erreurs. Nous allons maintenant présenter les problèmes les plus significatifs que nous avons dû résoudre.

Suite à la première lecture de l'énoncé, nous avons suivi les directives sans trop réfléchir. Bien que l'énoncé fût clair et précis, notre méconnaissance des mécanismes des classes abstraites et des interfaces a entraîné nos premières erreurs. Nous n'avions pas correctement géré l'héritage des paramètres et des méthodes dans le package « representation ». En effet, nous avions déclaré les mêmes attributs dans toutes les classes de Nœud, et n'avions pas réfléchi à ou coder quelles méthodes. Lorsque nous avons créé une première maquette d'histoire pour les tests, une instanciation peu réfléchie de nos nœuds nous a rapidement bloqués. La compréhension des mécanismes d'héritage de classes a été la clé pour résoudre ce problème, ce que nous avons effectivement fait. Cette réflexion nous a également permis de structurer un univers plus propre, bien que conscient du fait que ses mécanismes ne soient pas très complexes. Si nous devions recommencer, nous chercherions à complexifier notre univers de jeu pour offrir davantage d'enjeux et de réflexion à l'utilisateur.

Le deuxième problème significatif que nous avons rencontré concernait la combine des nœuds et du package « Univers ». Comment faire en sorte que les décisions influent sur les paramètres du jeu et vice versa ? Nous avons d'abord commencé à créer une fonction chgt (changement) dans notre classe Histoire, appelée dans notre boucle principale assurant le déroulement du jeu. Cette fonction prenait le « currentNode » et, selon son identifiant, effectuait les changements nécessaires. Le problème est survenu lorsque nous avons introduit nos ImageNode et SoundNode. En effet, ces événements ne provenant pas de la classe Node, ils ne possédaient pas d'identifiant. Nous avons donc transféré cette fonction à l'intérieur des classes DecisionNode et InnerNode. Nous avons également modifié la signature des méthodes chooseNext des différentes classes afin qu'elles retournent un Event plutôt qu'un Node. Ainsi, le jeu s'écoule maintenant comme prévu.

Enfin, nous avons évité une problématique grâce à ce que la communauté des programmeurs appelle « la magie de l'informatique ». Lors de la création de nos classes Loader et Saver, nous avons rencontré d'importants bugs dus à une mauvaise gestion des Scanner. Nous avions déjà rencontré des problèmes de gestion des Scanner dans notre méthode chooseNext, car notre première intuition avait été d'ouvrir et de fermer un nouveau Scanner à chaque appel de cette fonction. Sans trop nous poser de questions, nous avons déclaré un Scanner static dans la classe DecisionNode, afin d'en ouvrir un seul dans tout le programme, et les bugs ont disparu. Ce problème est réapparu lors de la création de nos classes Saver et Loader. Des coups de ChatGPT nous ont amenés à ouvrir des Scanner à certains endroits puis à d'autres non, tout en utilisant notre Scanner de la classe DecisionNode jusqu'à tout débugger. Cependant, un problème persistait : lors de l'appel de la méthode save() dans Saver, avant même que l'utilisateur ne puisse saisir le nom de sa partie pour l'enregistrer, le Scanner prenait une chaîne de caractères vide comme saisie, et donc la partie n'avait pas de nom à sa sauvegarde. Après de nombreux petits changements

aléatoires de code, nous avons remplacé s.nextLine() par s.next(), et comme par magie, le Scanner reconnaissait la saisie de l'utilisateur. Bien que nous souhaitions permettre à l'utilisateur de choisir un titre avec des espaces, nous avons dû y renoncer. Des recherches nous ont orientés vers des problèmes éventuels de concurrence de threads, bien que nous n'ayons pas encore compris clairement quel était le problème derrière l'appel de s.nextLine().

3. Retour sur le projet et conclusion

Nous avons pris plaisir à élaborer ce projet, offrant une compréhension et une mise en pratique très générale et complète du cours. La logique sous-jacente à la programmation orientée objet a été brillamment illustrée tout au long de ce projet, et nous l'avons trouvée passionnante. En tant qu'enthousiastes de la mise en pratique des nouvelles théories informatiques, ce projet nous a donné un avant-goût particulièrement savoureux de cette logique programmatique. L'écriture de l'histoire était divertissante et suffisamment concrète pour que nous puissions percevoir l'utilité de ce langage. La difficulté du projet était stimulante mais surmontable, ce qui était approprié. Cependant, certains éléments du projet nous ont moins enchantés. Nous n'avons pas particulièrement apprécié l'étape de mise en place des tests JUnit ainsi que la documentation Javadoc. Bien que nous comprenions l'utilité générale de ces deux fonctionnalités Java, leur pertinence ici nous a échappé. De plus, programmer ces deux fonctionnalités s'est révélé long et ennuyeux. Le manque de connaissances sur les threads et leur impact nous a également perturbés dans l'élaboration du sujet, car leur utilisation aurait été intéressante pour afficher des images, jouer des fichiers audios et bien gérer les objets de type « Scanner ». Finalement, nous avons trouvé que le travail était un peu trop étendu.

En conclusion, ce projet a été une opportunité passionnante pour mettre en pratique les concepts de programmation orientée objet. La conception réfléchie de la hiérarchie des classes et l'ajout d'éléments de gameplay ont enrichi l'expérience de jeu.

4. Sources

ChatGPT - https://chat.openai.com/
Stack overflow - https://stackoverflow.com/
Openclassroom - https://openclassrooms.com/fr/
w3schools https:- //www.w3schools.com/